計算機工学概論課題

計数工学科 4 年 30500 岡山友昭

2004 年 8 月 5 日

1　簡易ベクトル・行列演算ライブラリ

講義で配られた C プログラム、 matrixvector.c の骨組みを元に、改善や新たな実装を行った。主な部分を以下に示す。

1.1　行列の記憶領域確保と解放 (AllocMat, FreeMat)

行列の記憶領域確保 (AllocMat) は、例えば A が m × n 行列の場合には A[m] に 1 次元配列として m ∗ n 個分のデータを格納し、A[i] (i = 0, 1, ... m-1) は A[m][i ∗ n] を指すポインタとなっている。例えば 4 × 4 行列の場合は図 1 のようになっている。

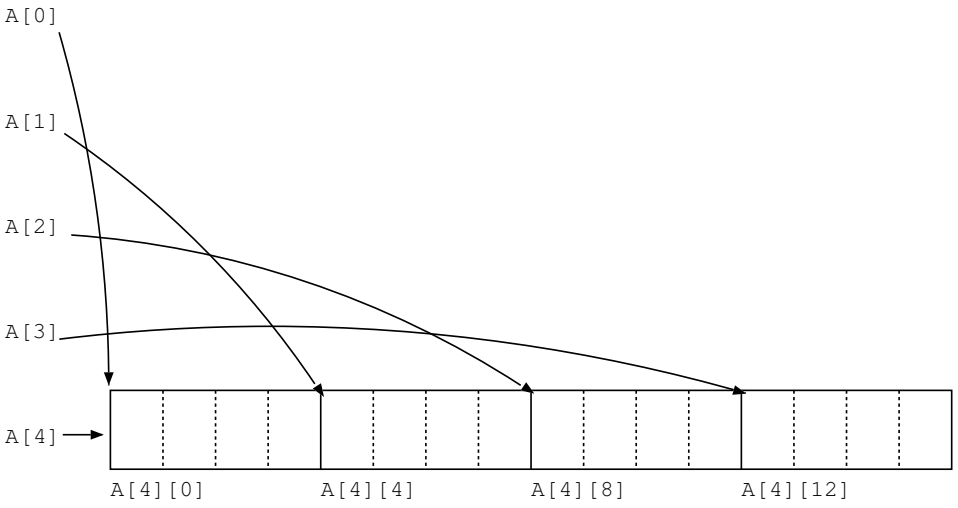

図 1　4 × 4 行列のデータ格納の様子

初めは A[0] にデータを格納していたのとは違い、 A[m] にデータを格納するようにしたため、この記憶領域を解放する FreeMat 関数には列のサイズを渡さなければならなくなったことに注意。

1.2 行列の行の入れ換え (SwapRowsInMat)

ガウスの消去法や LU 分解の際に必須な行の入れ換えをする関数、SwapRowsInMat は、初めは律義に全ての要素をとりかえていたが、せっかくデータの格納を先ほどのような形にしたので、その利点を生かしてポインタの入れ換えをするだけにした。これでここの操作のオーダーが m から 1 になった。これとは違い、列の入れ換えをする関数 SwapColsInMat は、まともに全ての要素をとりかえる操作をする (遅い)。

1.3 行列の無限大ノルム (InfNormMat)

行列の無限大ノルムを求める InfNormMat を実装した。ここで、$n \times n$ の実数を要素とする行列 $A = \{a_{ij}\}$ に対する最大値作用素ノルム $\|A\|_\infty$ は

$$\|A\|_\infty := \max_{\|\boldsymbol{x}\|_\infty \leq 1} \|A\boldsymbol{x}\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|$$

で与えられることを用いている。

1.4 LU 分解 (LUdecomp)

逆行列を求めるときや連立 1 次方程式を解く際にとても有効な手法、LU 分解のメソッド LUdecomp を実装した。この際に参考にしたサイトを最後 (参考文献) に挙げておく。これらのプログラムに比べて私の実装したものの良い点を以下に示す。

1. ピボット選択を行っていること (というかメソッド中ほとんどがこのコード)
2. ピボット選択の際、行のスケーリング (重みづけ) を [きちんと] していること
3. 行列式も求められること。ただし計算量は全く変わらない。
4. 行の入れ換え操作はオーダー 1 でできること (これは上で述べた)
5. 与えられた行列に対して破壊的な操作を行わないこと
6. LU 分解した情報を保持できる形で返していること

5. は、与えられた行列をコピーした新しい行列を用意してそれに操作すればいいものを、与えられた行列そのものに対して操作をしてしまうというプログラムがあったことを言っている。

6. については、LU 分解した情報を何度も使いまわせるということが LU 分解の利点であるはずなのに、その情報を捨ててしまうものが多かったということである。逆行列はそれでもガウス・ジョルダンの方法と計算量は変わらないが、連立 1 次方程式を解く際には LU 分解をしなければいけない分、計算量がガウスの消去法よりも増えてしまう。1 回連立 1 次方程式を解くだけならガウスの消去法で解く SolvLinEqGauss メソッドを使えば十分である。同じ行列を使って何回も問題を解く場合は、LUdecomp と以下の SolvLinEqLU を使うとよい。

1.5 LU 分解を利用して逆行列を求める (InvMatLU)

LU 分解した情報を利用して、逆行列を求めるメソッド InvMatLU を実装した。これを行う前に必ず LUdecomp を実行して、LU 分解済みの行列とピボット選択情報を InvMatLU に渡す必要がある。

1.6 LU 分解を利用して連立 1 次方程式を解く (SolvLinEqLU)

LU 分解した情報を利用して、連立 1 次方程式を解くメソッド SolvLinEqLU を実装した。これを行う前に必ず LUdecomp を実行して、LU 分解済みの行列とピボット選択情報を SolvLinEqLU に渡す必要がある。

2 連立 1 次方程式の精度保証つき計算

実装した簡易ベクトル・行列演算ライブラリ、matrixvector.c を使って、連立 1 次方程式を精度保証つきで計算するプログラム accuracy.c を実装した。理論については適当な文献参照のこと、ということで逃げて、メソッドについての簡単な説明にとどめる。ちなみに、ノルムは全て無限大ノルムで実装した。

2.1 誤差評価に必要なパラメータの値 (coefficient)

今回の場合、計算機で求めた A の逆行列を $\widetilde{A}^{-1}$ とすると、最大誤差評価をするためには

$$\|\widetilde{A}^{-1}A - I\| < 1$$

であることが条件になるので、$\|\widetilde{A}^{-1}A - I\|$ の値を調べる coefficient を実装した。実際は $\widetilde{A}^{-1}$ を使う必要はなく、$\|B^{-1}A - I\| < 1$ を満たす正則行列 B ならば何でもよいのだが、適当な B を探すことが難しいので今回は $\widetilde{A}^{-1}$ を用いた (ただし計算量はそれなりに必要)。

2.2 計算結果の最大誤差を求める (MaxError)

連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を計算機で求めた解を $\widetilde{\boldsymbol{x}}$ とすると、$\|B^{-1}A - I\| < 1$ を満たす適当な正則行列 B に対してその計算結果の誤差 $\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|$ は

$$\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\| \leq \frac{\|B^{-1}(A\widetilde{\boldsymbol{x}} - \boldsymbol{b})\|}{1 - \|I - B^{-1}A\|}$$

と評価されるので、その最大値を求める MaxError を実装した。B^{-1} は実際には上と同じく $\widetilde{A}^{-1}$ を用いた。

3 テスト計算プログラム (calculate.c)

乱数で作った連立 1 次方程式を精度保証つきで計算するテストプログラム、calculate.c を実装した。乱数の発生には、C 標準関数 rand() よりも様々な良い点があるとされる Mersenne Twister[4] を用いている。使い方は make したあと ./calculate を実行するだけである。

1 回目の問題は乱数で発生させた行列を用いるが、2 回目は性質の悪い行列にするために意図的な操作を加えた行列を用いている。1 回目と 2 回目を比べると精度がだいぶ違うことがわかると思われる。calculate.c の行列のサイズ (DIM) を 1000 などの大きな数にして試す場合は、出力命令 (prinf) の部分はコメントアウトすることをおすすめする。

4 アーカイブ一覧

matrix.tar.gz の中身を記す。

- Makefile
- accuracy.c
- accuracy.h
- calculate.c
- matrix.pdf
- matrixvector.c
- matrixvector.h
- mt19937ar.c
- mt19937ar.h

参考文献

[1] 連立一次方程式
http://www.fuka.info.waseda.ac.jp/~kozo/suuchi/simple_equation/simple_equation.html

[2] C 言語によるアルゴリズム (コメント付き)
http://www.sra.co.jp/people/miyata/algorithm/

[3] Gauss の消去法と LU 分解
http://daisy.math.sci.ehime-u.ac.jp/users/tsuchiya/math/linear/gauss/index.html

[4] Mersenne Twister Home Page
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html